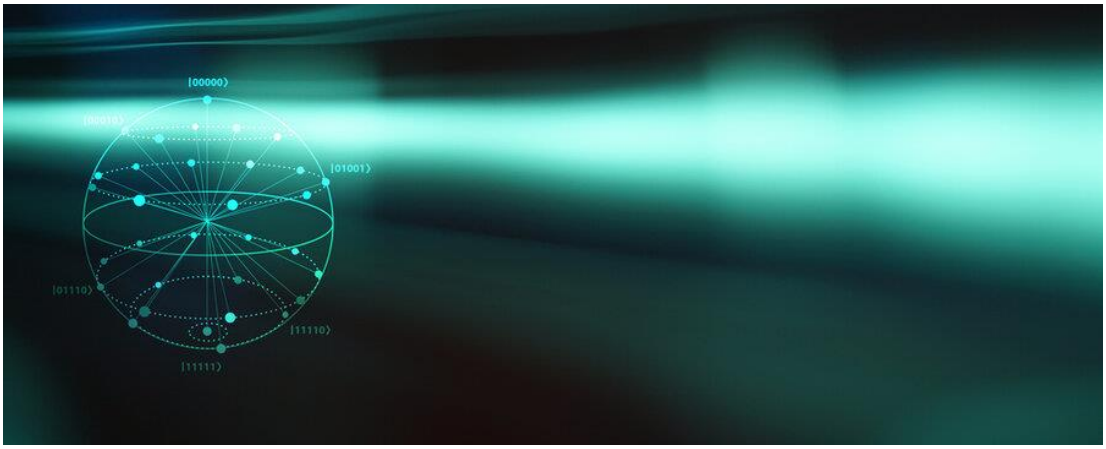
OpenQASM Debugger Visual Studio Extension Documentation

V0.4 09/04/2024



QUANTAG IT SOLUTIONS

Contents

Installation

Extension can be downloaded and installed from Microsoft Visual Studio Marketplace
https://marketplace.visualstudio.com/items?itemName=QuantagITSolutionsGmbH.openqasm-debug

Description

This VS Code extension connects to QSCore (Quantag Studio Core) component which runs instance of Quantum Virtual Machine (QVM) using DAP (Microsoft Adapter Debugger) protocol and allows:

- debug Quantum circuits written on OpenQASM or Python using Qiskit or PyTKET frameworks.
- perform step debugging on remote Quantum Virtual Machine
- see quantum state on every step
- see compiled OpenQASM code for python circuits in Disassembly View
- see visualized Quantum Circuits directly in VS Code and on web frontend.

The main idea is to establish a quantum development environment in a way like a classic software development environment with sources, assembler code, and direct hardware execution possibility.

How does it work?

After installing extension from Marketplace, it registers itself for .py and .qasm files. *There can be other extensions also installed and registered for same file types*. They can work together; just be sure you choose the right menu items and command for every extension.

Before any action sources from local workplace must be submitted to cloud server where QVM working. To do it we do not use SSH Tunnels which is common way of working with remote file system from VS code, instead we submit all workplace files to server using own microservice. All files submitted after opening new Workplace to avoid delays when browsing files. So be careful with private data, do not open private data with activated extension.

When a user chooses '**Debug circuit'** with python file – source is analyzed on server and *updated* to add OpenQASM-export and rendering circuit commands depends on detected framework.

When choosing '**Debug circuit'** on OpenQASM file – this file imported to Qiskit on server using import-qasm commands and rendered to circuit image which later can be opened using command palette with command 'Render OpenQASM/Qiskit circuit'.

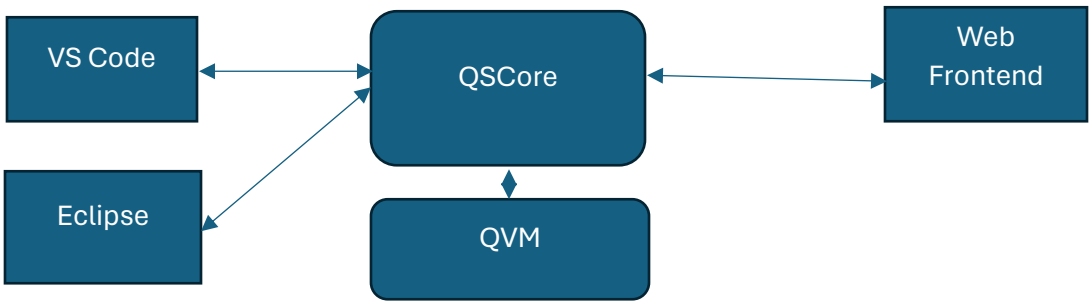
For TKET python sources – circuit generated to HTML and opened in external browser using 'Render TKET circuit.'

Rendered circuits and compiled OpenQASM files transferred from server to VS code using dedicated microservice.

Launching circuit on remote hardware implemented using adding commands to 'Command Palette'.

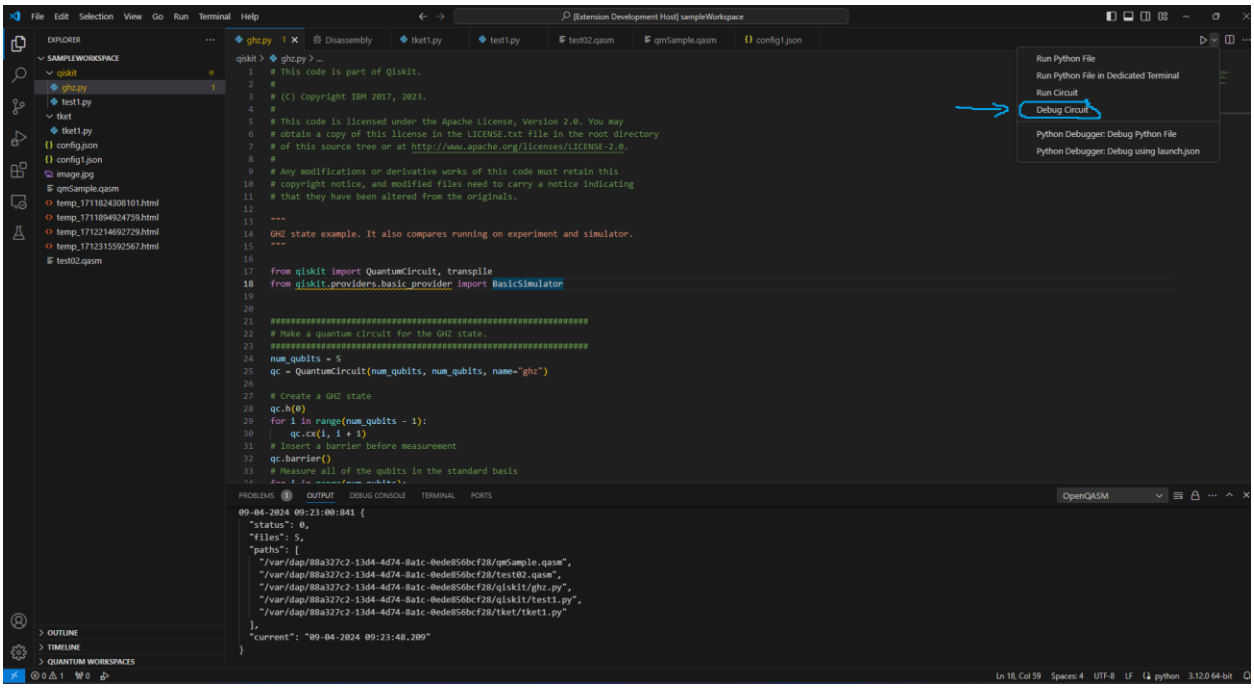
Hardware configuration for launching taken from workplace root file *'config.json'* which contains all information required for launch.

QSCore uses DAP protocol to communicate with VS Code, so it can work with any IDE which supports DAP, for example Microsoft Visual Studio, Eclipse or Emacs. Full list of supported IDEs can be found here: https://microsoft.github.io/debug-adapter-protocol/implementors/tools/



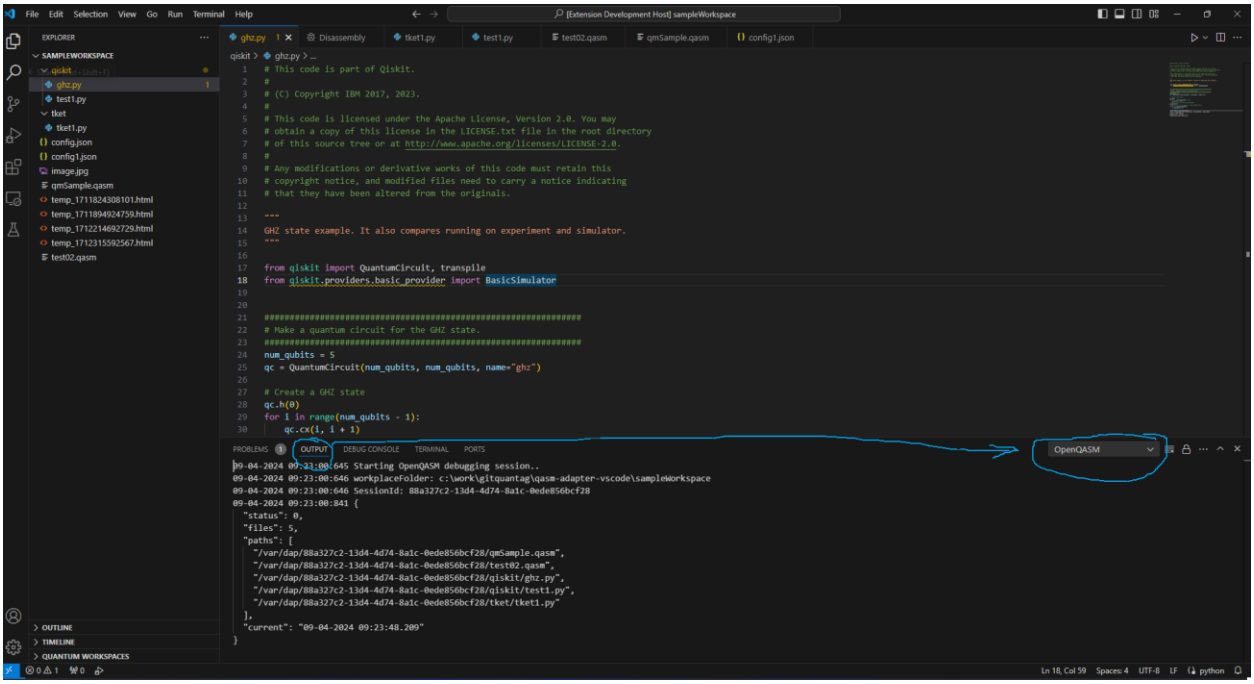
Activation and starting

Extension activated on .qasm and .py files and can be started by choosing ‘Debug Circuit’ command from ‘Run’ menu



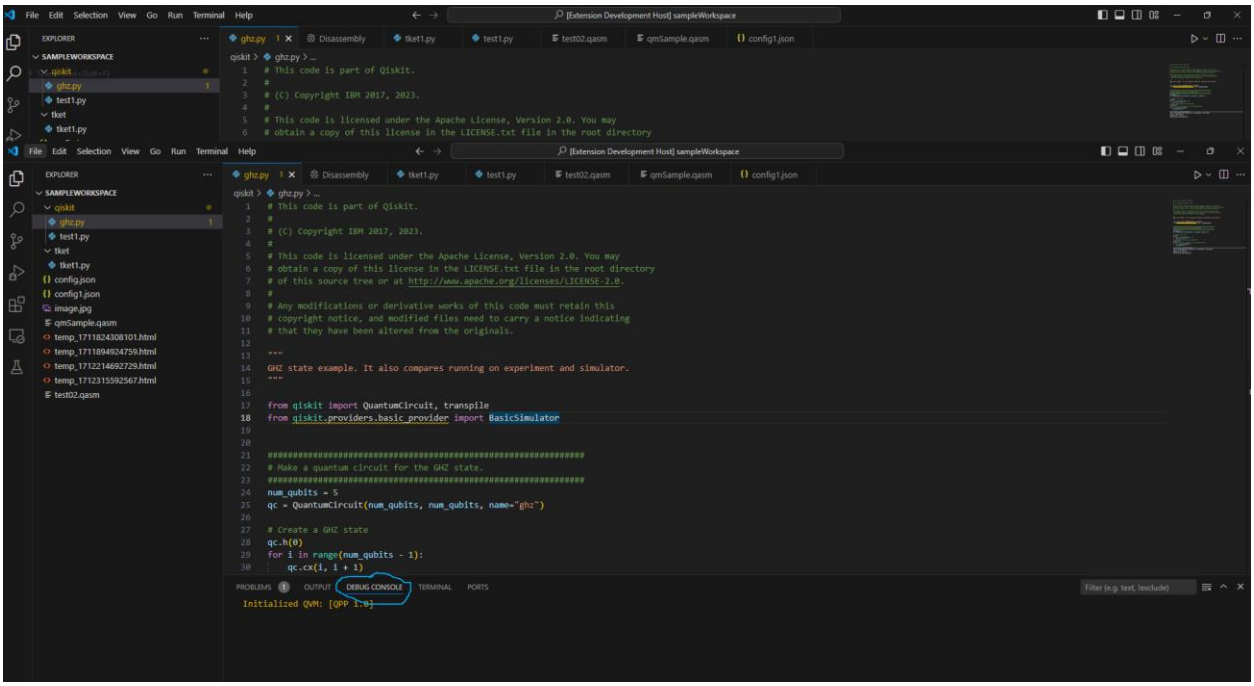
Output and debug messages.

After starting debugging session Extension debugging information can be seen in ‘Output’ View after choosing ‘OpenQASM’ output channel. *These messages are needed to debug the extension itself, not python or OpenQASM program*.



Debug console.

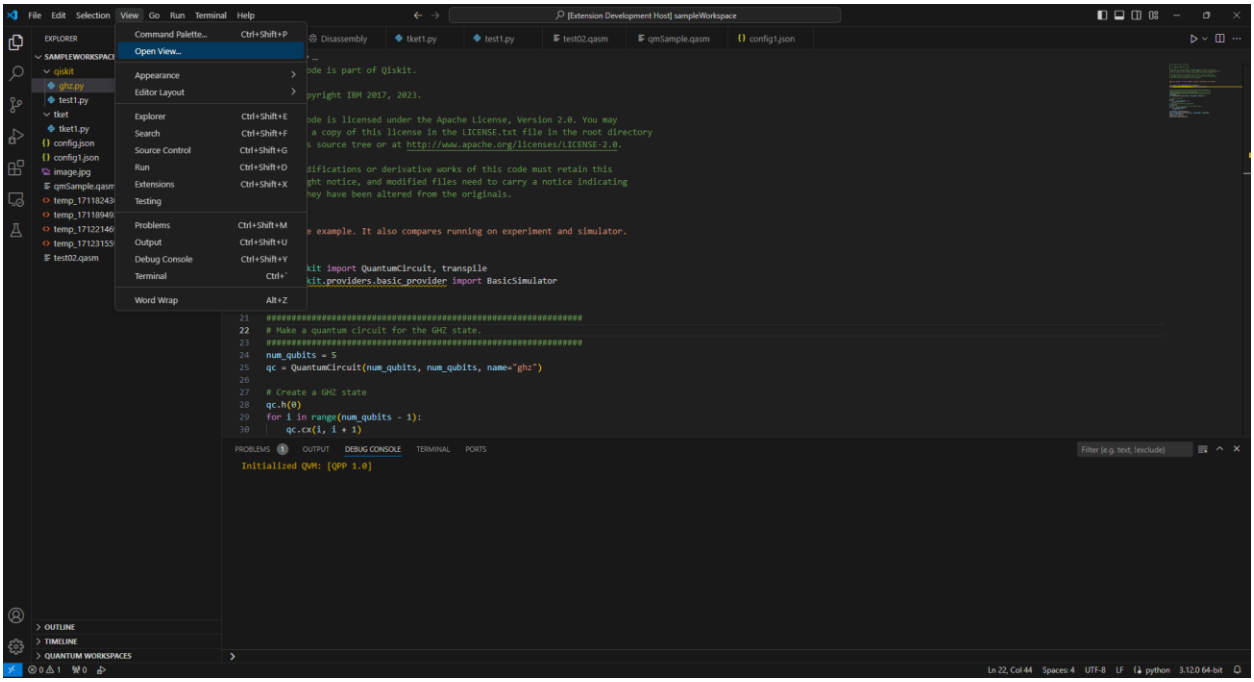
In Debug Console view you can see messages sent from *remote* Quantum Virtual Machine. For example, OpenQASM parsing errors. These messages sent using OutputEvent DAP command.



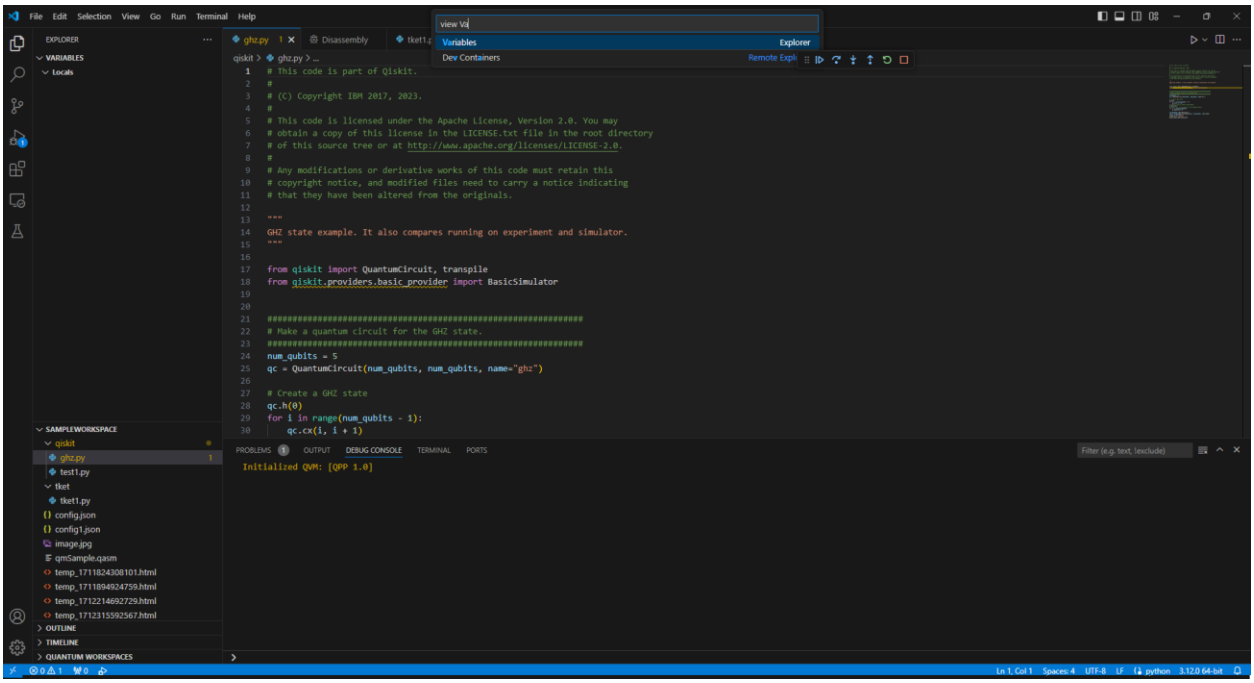
Quantum state

Quantum state of the QVM can be seen in view ‘Variables’.

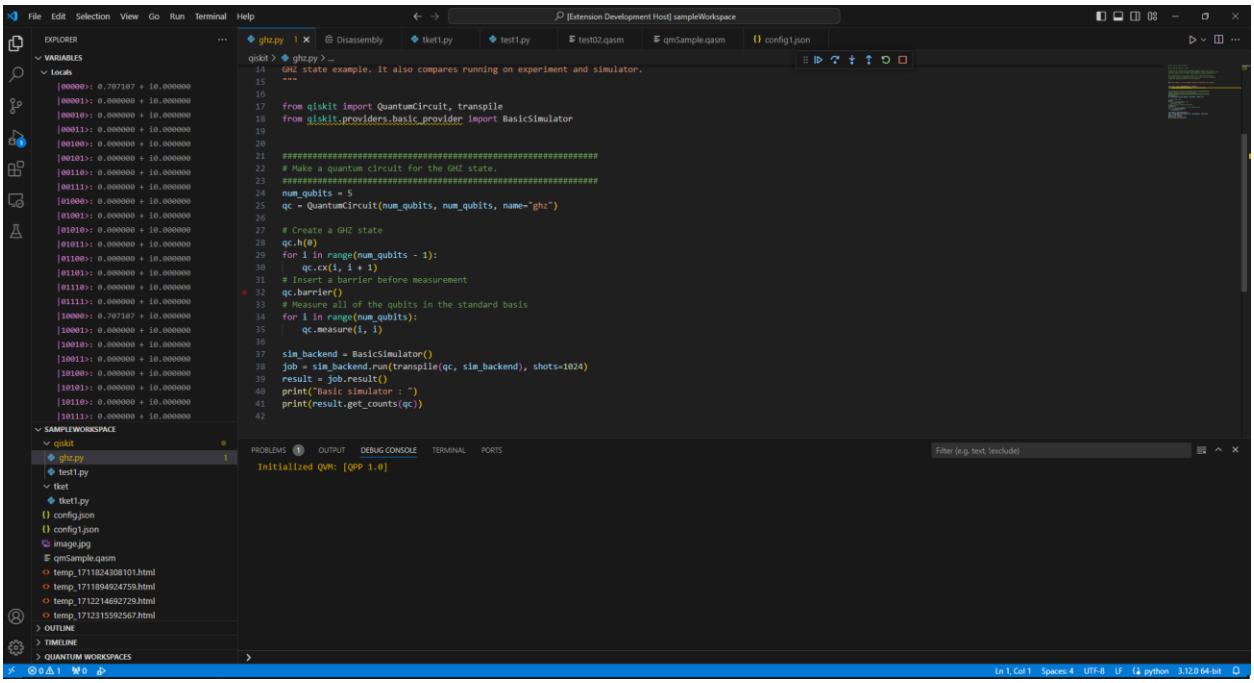
After starting debugging session go to ‘View’ menu and choose ‘Open View..’ item.



Then type ‘Variables’ in opened prompt



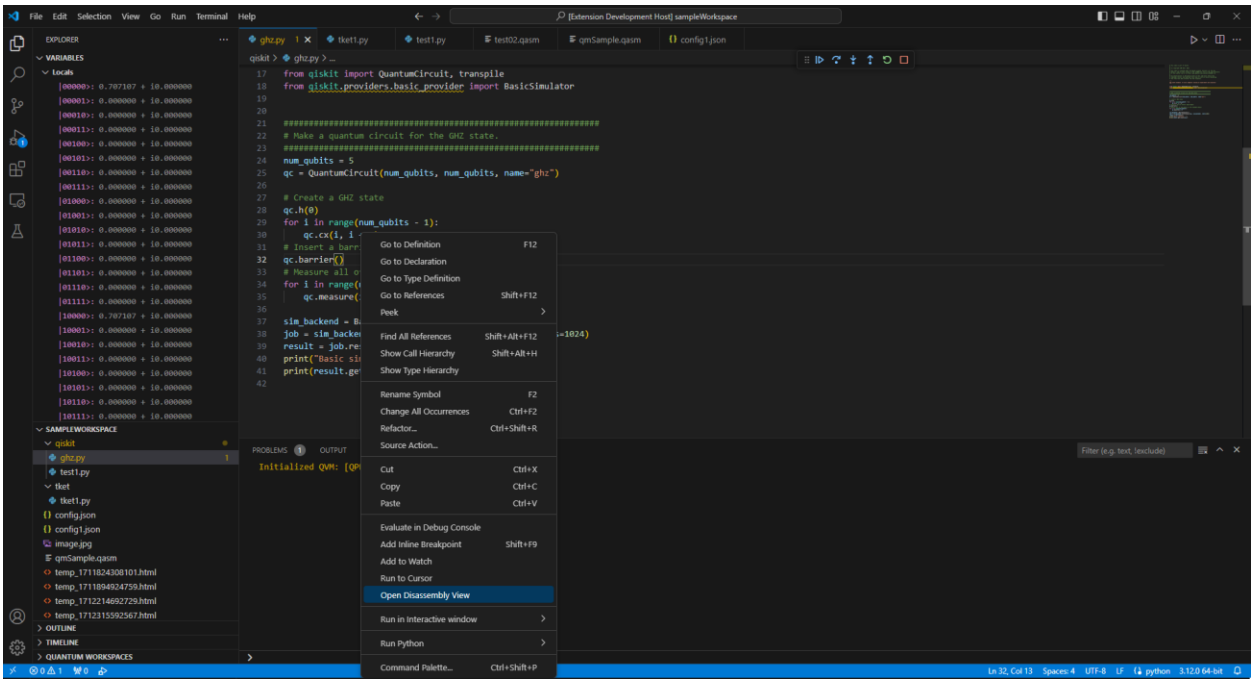
Quantum state can be seen in Variables View after parsing circuit after first ‘Step’ command of debugger.



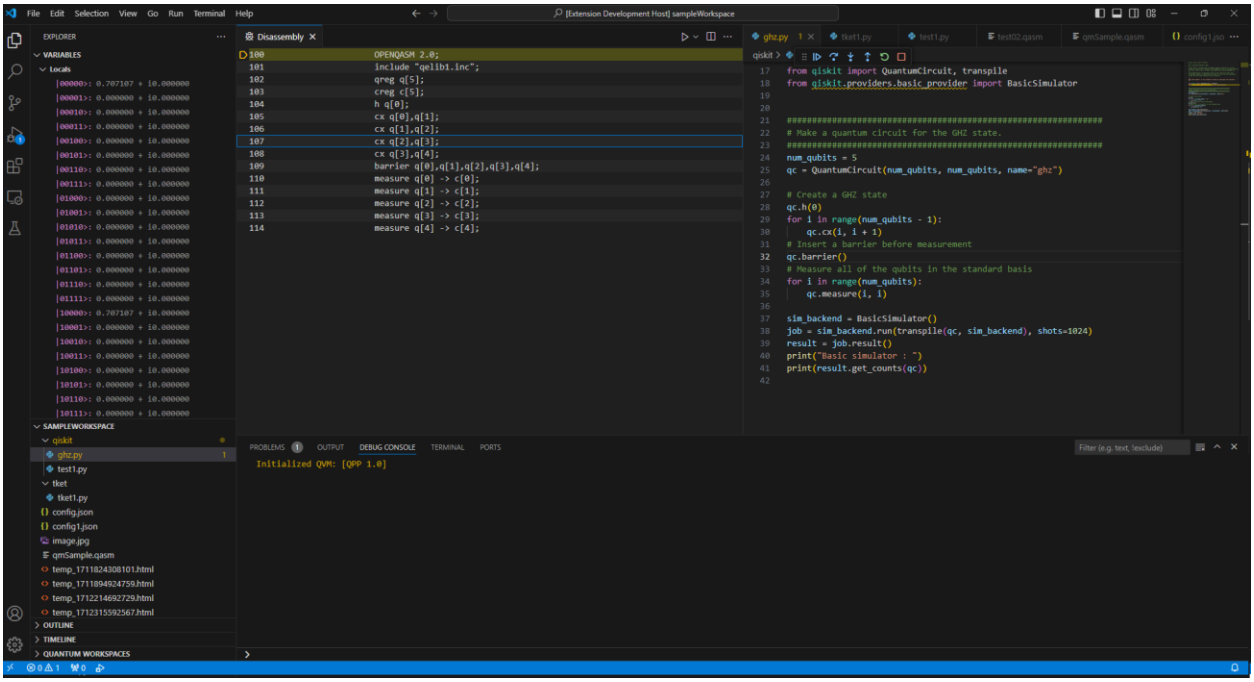
For example, line **|10000> = 0.707107 + i0.0** means that during measurement probability to get qubit #0 in state ‘1’ and qubits #1-#4 in state ‘0’ equals ½. (0.707107 ^ 2). More about state of quantum system of N qubits can be found here.

Complied OpenQASM code

After the first ‘Step debug’ command of python script you can see OpenQASM code in ‘Disassembly View’ *if python code was compiled correctly to OpenQASM in QVM*.

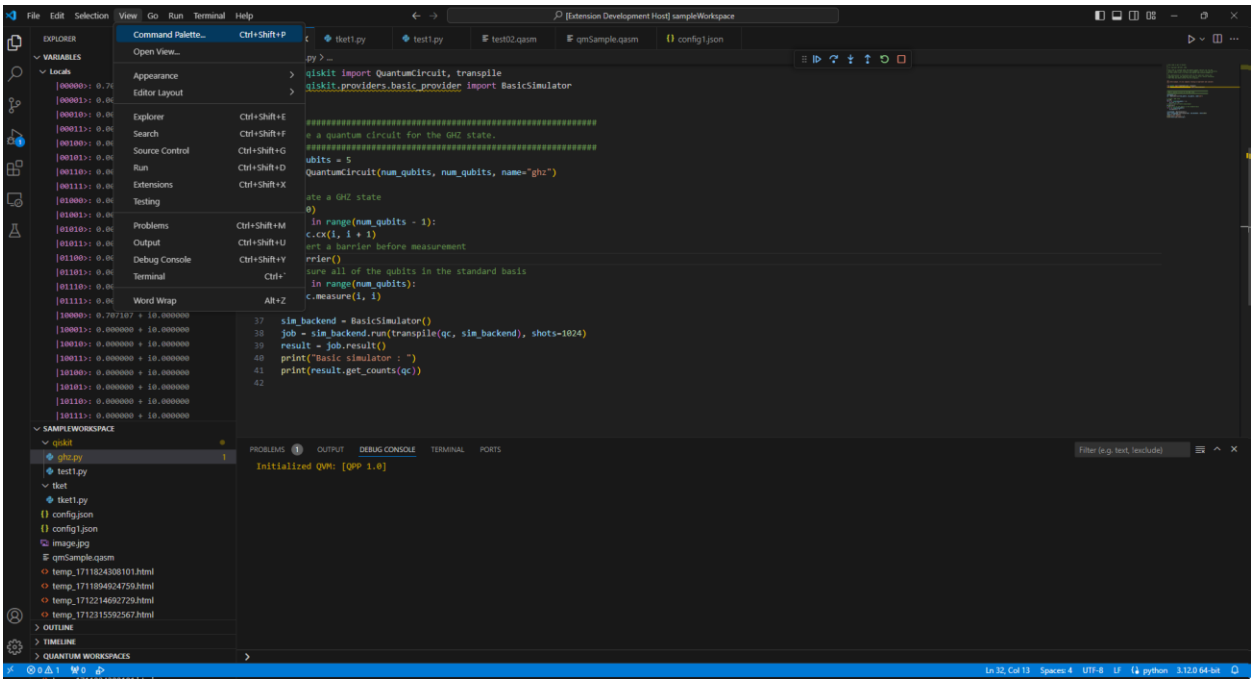


Disassembly View can be moved as any other View in VS code for convenience.



Command Palette

Extension exposes few commands which are available in Command Palette. To open Command Palette press ‘Ctrl+Shift+P’ in Windows or open menu ‘View’ and then ‘Command Palette..’ item

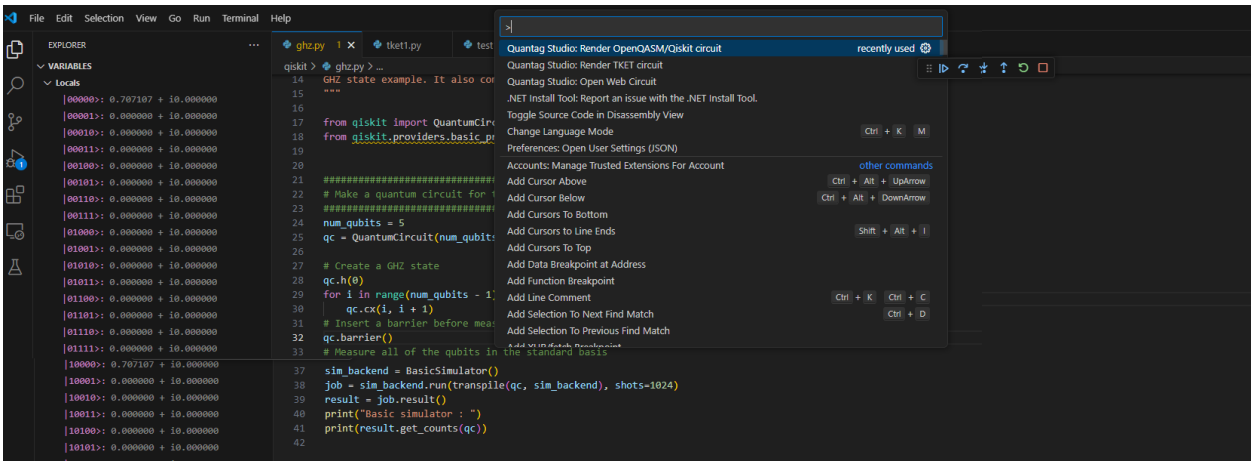


Some commands are available only during debugging sessions, some only for ‘.py’ files or only for ‘.qasm’ files. If you can not find some command check that you are in the right mode and correct file is opened.

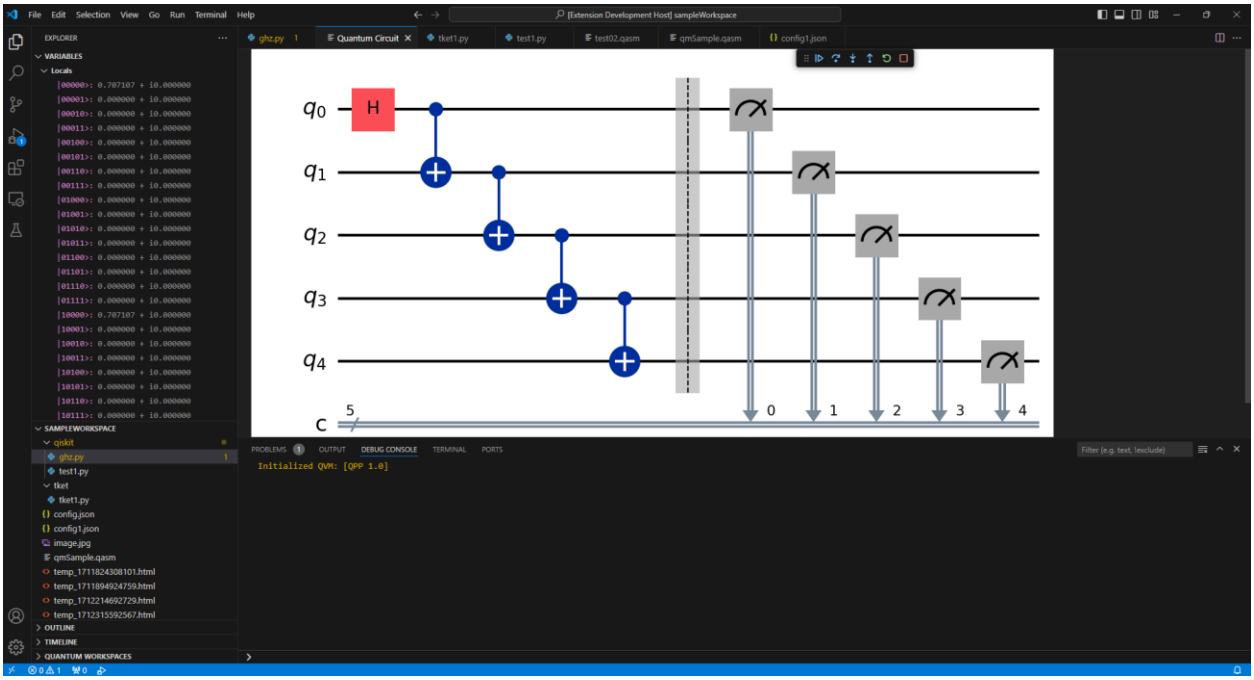
Render OpenQASM/Qiskit circuit command

This command can be used during debugging session to render OpenQASM or Qiskit circuit within VS Code.

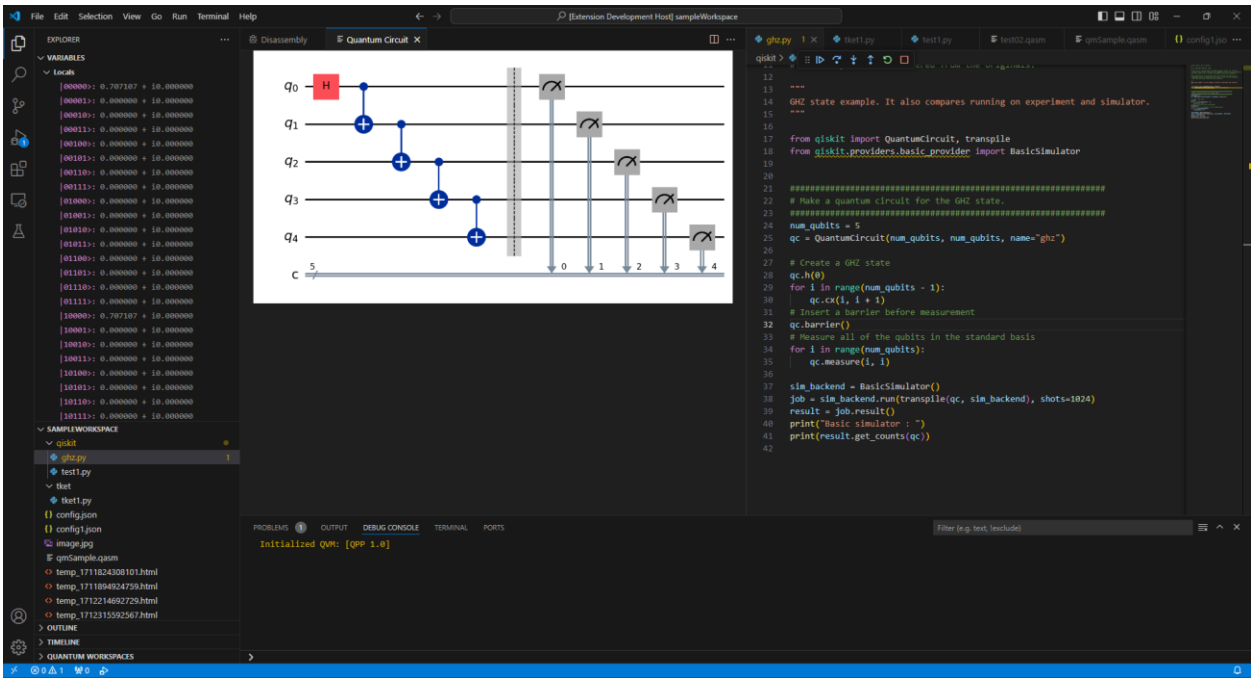
Open ‘Commands Palette’ and choose ‘Quantag Studio: Render OpenQASM/Qiskit circuit’



If circuit correctly parsed in QVM you will see rendered circuit in new View in VS Code.



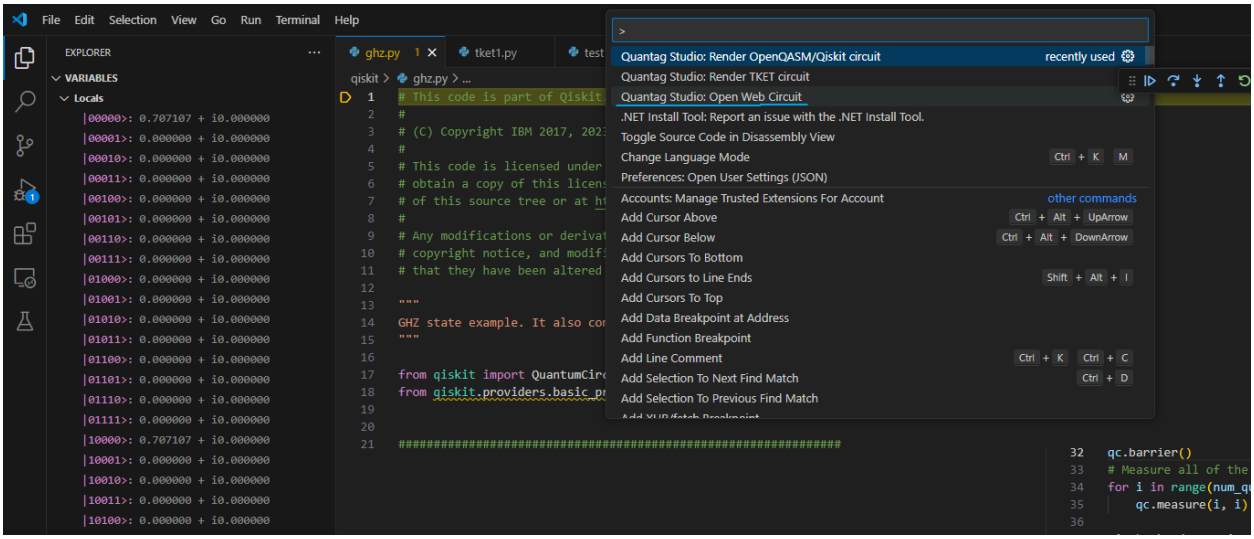
You can rearrange visualized circuit like any other View in VS Code to see source code and circuit at same time:



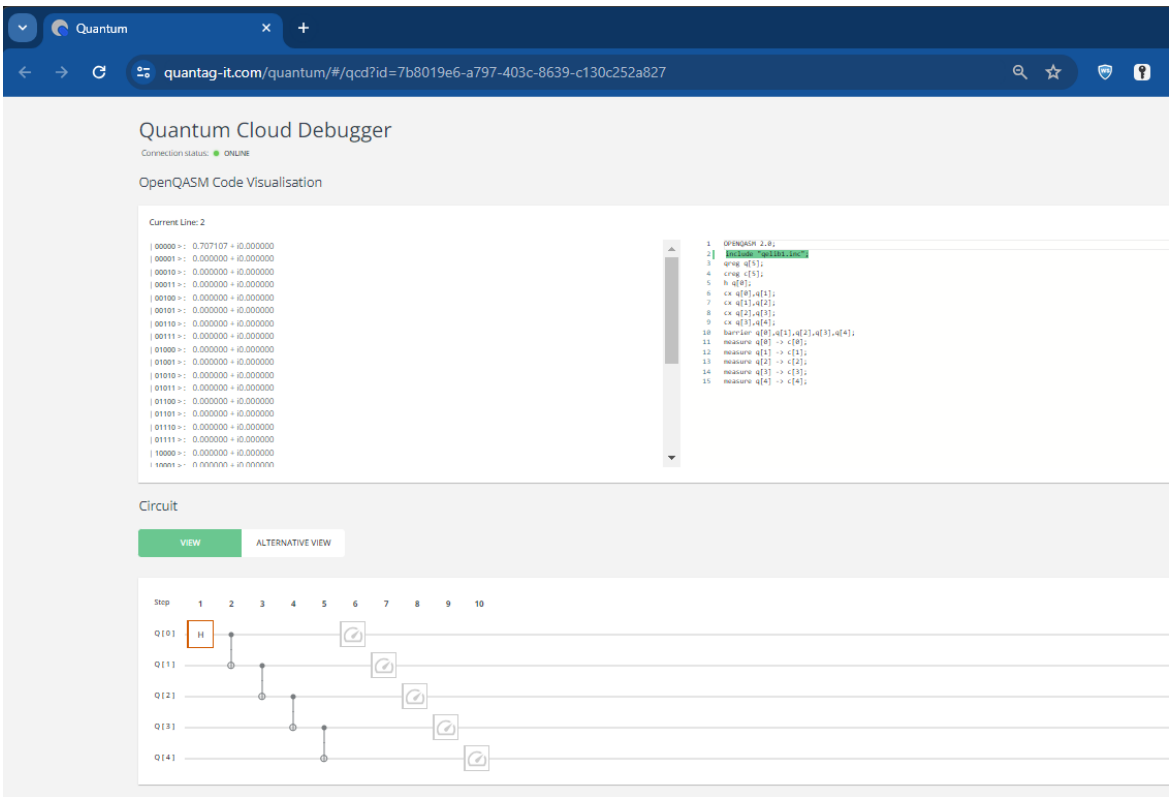
Web frontend

Additionally, QVM has its own web frontend which opens in external browser *after starting debugging session* and can render circuits, show quantum state and source. It is implemented on angular.js and communicates with QSCore using web-socket connection.

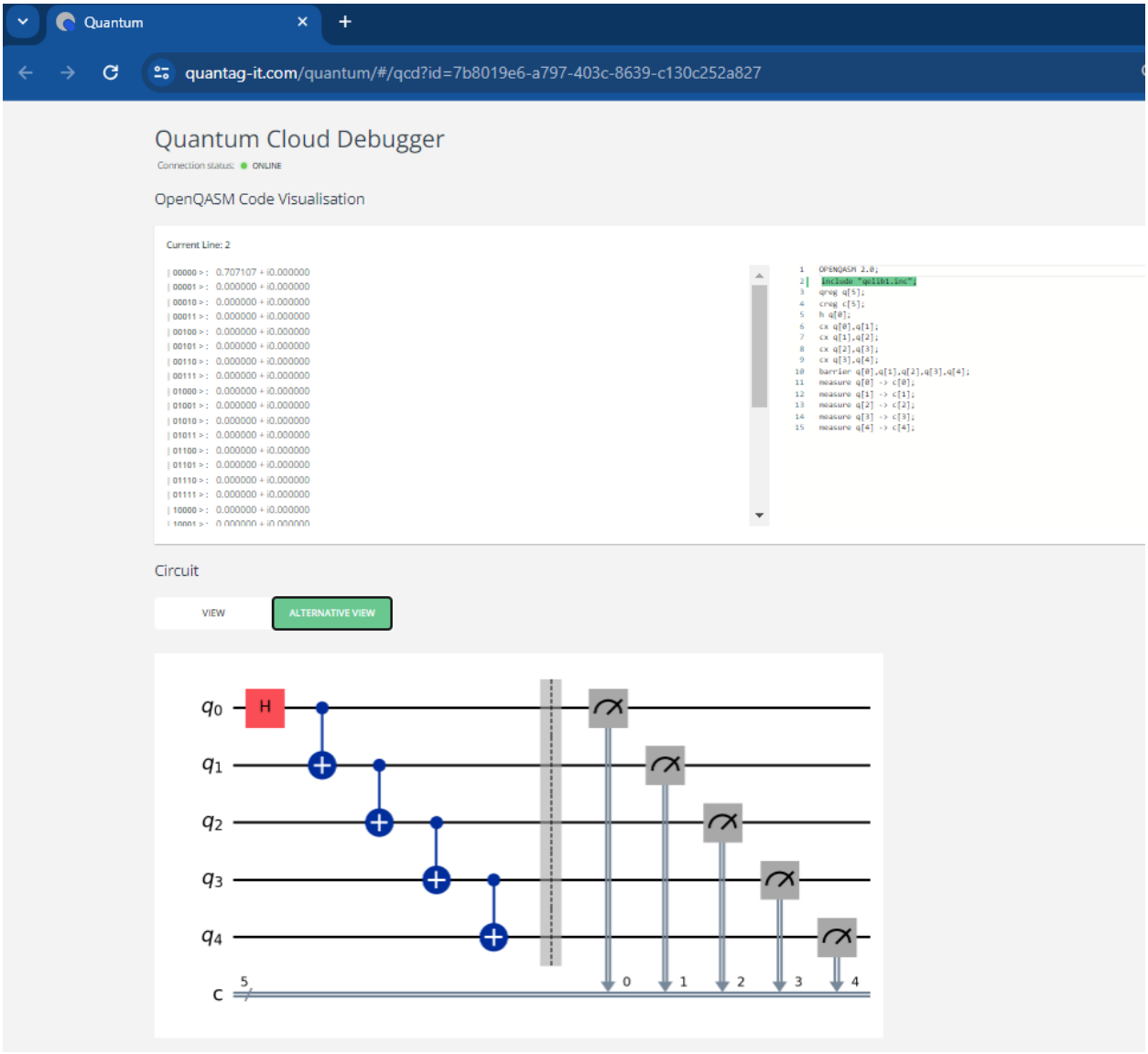
To open web frontend use command ‘Open Web Circuit’.



It can render circuits using own renderer in View or Qiskit renderer in ‘Alternative View’ is available.



Alternative Web View



Web frontend can be used to integrate QSCore in existing web cloud services.

Limitations and future plans

QSCore component and extension currently under active development. Not all features are fully implemented and stable.

Here is list of main limitations:

- not all circuits are working and supported. QSCore analyzes python sources and modifies them on the fly to add rendering and OpenQASM exporting. If circuit is complex with mixed functions and modules – it might fail, and errors will be shown in 'Debug output' view
- For big workplaces file transfer might take some time, and there can be delay at the beginning of session.
- QVM initialization can be slow for big number of qubits in circuit.

Future plans are:

- Adding commands for Circuit Compression and Qubit-to-Qudits circuit conversions
- Adding support of new hardware for direct execution
- Adding support of cloud-based execution of circuits
- Adding different types of QVM with hardware acceleration (Nvidia cuQuantum https://developer.nvidia.com/cuquantum-sdk) currently we use https://github.com/softwareQinc/qpp as QVM.
- Adding support of hybrid programs (classic + quantum)
- Adding visualization of measurement results in VS code
- Adding circuit step animation of debugging session in VS Code.

Please report found bugs to support@quantag-it.com